Patrons de conception

Jean-Marc Jézéquel

Les patrons de conception

Jean-Marc Jézéquel

Mots-clés : *design patterns, patron de conception, conception orientée objet, schémas d'interactions*.

Résumé : la définition de la nature de l'activité de conception de logiciel est restée longtemps assez floue. La généralisation dans l'industrie d'une approche par objets de la conception de logiciel a permis de focaliser la tâche de conception sur la description de schémas (ou motifs) d'interactions entre objets, et d'assigner des responsabilités à des objets individuels de telle sorte que le système résultant de leur composition ait les propriétés voulues de flexibilité, maintenabilité, efficacité, etc. L'aspect récurrent de certains de ces schémas d'interactions entre objets a conduit à vouloir les cataloguer : on les appelle alors « *design patterns* ». On choisit ici de traduire cette notion par «patron de conception», d'une part pour sa sonorité proche de l'anglais mais aussi pour mettre en évidence le parallèle avec la notion de forme prédéfinie existant dans un patron en couture qui laisse en pratique de nombreux degrés de libertés dans son utilisation. Perçus depuis les dix dernières années comme des techniques essentielles à la conception objet, les patrons de conception ont fait l'objet d'un véritable engouement, tant dans l'industrie que dans le monde de la recherche, et ont donné lieu à de multiples publications, tant de catalogues que d'articles scientifiques les analysant. Ce chapitre a pour objectif de familiariser le lecteur avec cette notion de « patron de conception », ainsi que de montrer son utilisation dans un processus de conception de logiciels à objets.

1. Introduction

1.1. Définition simple

La notion de « *design pattern* », qu'on choisit ici de traduire par « patron de conception », a été popularisée au milieu des années 90 et a depuis profondément marqué la pratique de la conception de logiciels orientés objets. Qu'est-ce qu'un « patron de conception » ? Une définition simple et communément admise en fait « une solution à un problème de conception dans un contexte ». Ceci mérite bien sûr quelques explications complémentaires :

- la notion de contexte se rapporte à un ensemble de situations récurrentes dans lesquelles le patron de conception peut s'appliquer ;
- le problème fait référence à un ensemble de forces, à la fois objectifs et contraintes, qui se produisent dans ce contexte ;
- la solution fait référence à une forme ou une règle de conception canonique qui peut être appliquée pour résoudre ces forces.

1.2. Historique et motivations

L'idée d'identifier et de documenter systématiquement les patrons de conception en tant qu'entités autonomes clés pour la conception de logiciels date de la fin des années 80. Elle a été popularisée par des gens comme Beck, Ward, Coplien, Booch, Kerth, Johnson, etc. (connus sous le nom de « Hillside Group »), mais il est clair que la contribution séminale à ce domaine a été la publication en 1995 du livre *Design Patterns: Elements of Reusable Object Oriented Software* par la désormais célèbre « bande des quatre » (*Gang of Four*, ou GoF) : Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides (Gamma 1995). Ce livre, l'un des plus vendus de toute l'histoire de l'informatique, a constitué le premier catalogue de patrons de conception applicable à une grande variété d'applications. Dans ce premier catalogue, la plupart des patrons de conception se focalisent sur un aspect particulier de la conception de logiciel, à savoir sa flexibilité. Il s'agit donc pour l'essentiel de solutions de conception visant à rendre le logiciel conçu plus évolutif, que ce soit dynamiquement ou tout simplement pour faciliter de futures opérations de maintenance. L'idée clé des patrons de conception du GoF est de permettre de faire

varier certains aspects de la structure du système indépendamment des autres aspects, afin de rendre le système robuste à certains types de changements qui peuvent alors être effectués sans reconception.

Depuis ce premier opus, de nombreux autres catalogues sont apparus, par exemple les quatre volumes de PloPD (*Pattern Languages of Program Design*, voir (Coplien 1995, Vlissides 1996, Martin 1998, Harrison 2000)) ou encore le catalogue en ligne (Tichy 2003)). Certains de ces catalogues contiennent des patrons de conception qui prolongent la tradition du GoF, parfois en se concentrant sur des patrons de conception spécifique d'un métier particulier (voir l'excellent chapitre sur les patrons de conception pour les télécommunications dans (Vlissides 1996) ; mais d'autres en élargissent le spectre pour aborder une plus grande variété d'aspects extra-fonctionnels de la conception, comme la fiabilité, la prise en compte de la répartition des données ou des calculs, la ponctualité, la sécurité, la mise à l'échelle, les performances etc. D'autres élargissent encore ce spectre en abordant des patrons qui ne sont plus de conception, mais par exemple d'analyse ou de rétroconception (Demeyer 2003) ou même encore hors du domaine du logiciel, avec des patrons d'organisation (Coplien 2005).

Aujourd'hui, il est indéniable que les patrons de conception sont largement acceptés comme des outils particulièrement utiles pour guider et documenter la conception de logiciels à objets. Mais au-delà de cet aspect de base, les patrons de conception jouent plusieurs autres rôles dans le processus de développement :

- ils fournissent tout d'abord un *vocabulaire partagé* pour communiquer à propos de conception logicielle ;
- grâce à ce vocabulaire, des constructions complexes peuvent être décrites avec un meilleur niveau d'abstraction, ce qui permet de *maîtriser la complexité globale* d'un système ;
- finalement, les patrons de conception constituent une base d'expérience pour la construction de logiciels sous forme de *briques de base de conception* à partir desquels des conceptions plus complexes peuvent être élaborées.

1.3. Points clés

1.3.1. Partage de savoir faire de conception

Les patrons de conception représentent des solutions à des problèmes qui se produisent de manière récurrente lorsqu'on développe du logiciel dans un contexte particulier. Ils peuvent être considérés comme des micro-architectures réutilisables qui participent de l'architecture globale du système : ils capturent les structures statiques et dynamiques des collaborations entre les éléments clés d'une conception par objets, à savoir classes, objets, méthodes, messages, etc.

Parce que les patrons de conception identifient et documentent l'essence de solutions pertinentes à des problèmes de conception, ils sont d'une certaine façon assez proches de la notion d'idiomes, ou d'astuces de codage, qui existent pour la plupart des langages de programmation (voir par exemple (Coplien 1992) pour des idiomes C++). La principale différence est que les patrons de conception fonctionnent au niveau de la *conception* plutôt qu'au niveau de *l'implantation*, mais il est clair qu'ils capturent de manière similaire un certain savoir-faire, qui par définition distingue les concepteurs expérimentés des débutants.

Même si des brillants concepteurs de logiciels n'ont pas attendu la popularisation des patrons de conception pour produire des éléments de conception élégants, leur isolement face à l'ampleur de la tâche de conception des logiciels modernes a souvent rendu fragile leur contribution. Ce qui est singulier et si intéressant dans l'idée des patrons de conception est de surmonter cet isolement par un effort concerté au plan international pour identifier, documenter et classifier les meilleures pratiques de la conception orientée objet.

A cet égard, il convient de remarquer que la notion de patrons de conception n'est pas intrinsèquement liée aux technologies à objets. Mais force est de constater que, poussé par l'adoption massive par l'industrie à partir du milieu des années 90, le principal effort de catalogage a été effectué dans ce contexte, ce qui a permis d'obtenir effectivement un très grand retour sur investissement vis-à-vis de la qualité de conception des logiciels orientés objets modernes.

La principale contribution des patrons de conception est donc de capturer explicitement le savoir-faire des experts ainsi que les compromis de conception, et de cette façon permettre le partage d'un savoir-faire architectural entre développeurs. Ainsi la conception dans un projet n'est plus dépendante d'un seul gourou qui aurait tout dans la tête : le savoir-faire de conception d'une organisation peut être documenté à l'aide de patrons de conception. Ceci permet en particulier la répétition de succès antérieurs sur de nouveaux projets ayant le même type de problèmes de conception.

1.3.2. Documentation de conception

Les patrons de conception permettent aussi de documenter de manière concise l'architecture d'un système logiciel par la définition d'un vocabulaire

approprié. Les patrons de conception constituent donc les briques de base d'une architecture, et permettent ainsi sa compréhension à un plus haut degré d'abstraction, ce qui en réduit la complexité apparente. Plutôt que de penser la conception en termes de classes individuelles et de leurs comportements, il est maintenant possible de penser la conception en termes de groupes de classes collaborant selon un certain schéma prédéfini. Par exemple, pour expliquer une idée de conception sans faire référence aux patrons de conception, on devra dire à ses collègues que :

«Cette conception fait collaborer cet ensemble de classes ayant telles et telles relations entre elles de manière à ce que lorsque un événement se produit dans un objet instance de cette classe-ci, une certaine méthode est appelée, qui devrait déclencher telle et telle autre méthode sur les instances des sous-classes de cette autre classe ; lesquelles méthodes doivent rappeler l'objet source de l'événement pour obtenir la formation voulue.»

A l'inverse, un concepteur de logiciels familiarisé avec les patrons de conception les plus courants pourra simplement dire :

«Dans cette conception, cet ensemble de classes collaborent selon le patron de conception Observateur.*»*

Si ses collègues sont comme lui familiarisés avec ce patron de conception (qui sera d'ailleurs décrit plus en détail ci-dessous), ils comprendront immédiatement les structures statiques et dynamiques mises en jeu dans la collaboration entre classes de cette partie de l'architecture logicielle. De manière au moins aussi importante, les compromis de conception seront immédiatement explicités entre les principes de simplicité, d'efficacité, et la prise en compte d'autres forces extra-fonctionnelles comme la réutilisabilité, la portabilité, l'extensibilité, etc. Dans les phases initiales de la conception il est d'ailleurs souvent suffisant de savoir qu'on va utiliser un certain patron de conception à un certain point de l'architecture. Les détails concernant la manière dont ce patron de conception peut-être finalement implanté pourront être abordés plus tard.

1.3.3. Intérêt pédagogique

Finalement, les patrons de conception présentent un intérêt pédagogique certain. Il est souvent affirmé que le but des patrons de conception est d'apprendre par l'expérience, de préférence celle d'un autre. Codifier les meilleures pratiques de conception aide à disséminer cette expérience et aide à éviter certains des pièges du développement de logiciels. Parce qu'ils s'appuient sur les meilleures pratiques des technologies objets, les patrons de conception servent aussi à montrer comment utiliser élégamment et efficacement ces technologies, ce qui en a clairement permis une bien meilleure diffusion vers la fin des années 90.

2. Comprendre la notion de patron de conception

2.1 Connexion avec d'autres domaines

L'origine des patrons de conception se situe dans le domaine de l'architecture (celle du génie civil) avec les travaux de Christopher Alexander (Alexander 1979) : « *Chaque patron décrit un problème qui se produit encore et encore dans notre environnement, et ensuite décrit le coeur d'une solution à ce problème de manière à ce qu'on puisse utiliser cette solution plus d'un million de fois, sans jamais le faire deux fois exactement de la même manière. »*

Zhao et Coplien ont remarqué dans (Zhao 2003) la connexion entre la notion de patron de conception et celle de *symétrie* en mathématiques, qui est une opération sur un objet mathématique qui laisse cette objet inchangé (par exemple, une rotation ou une translation sont des exemples de symétries). Comme un patron de conception, une symétrie a la propriété essentielle de permettre le changement tout en préservant certains aspects inchangés.

Mais la notion sous-jacente aux patrons de conception est aussi connexe à la notion de savoir-faire caractéristique des métiers d'artisanat traditionnel. Elle englobe en particulier l'idée d'un plus haut niveau de compréhension élaboré à partir d'interactions entre composants primaires. On a ainsi pu tracer un parallèle intéressant entre l'apprentissage de la conception de logiciels et l'apprentissage du jeu d'échecs, où trois niveaux de maîtrise sont bien identifiés.

Au premier niveau il s'agit d'apprendre les règles de base : pour les échecs le nom des pièces, les mouvements légaux, l'orientation et la géométrie de l'échiquier, etc. ; et pour le logiciel la connaissance des algorithmes de base, des structures de données et des langages de programmation.
Au second niveau, il s'agit d'apprendre les principes : aux échecs la valeur relative de certaines pièces (comme la reine), la valeur stratégique des cases centrales de l'échiquier, pour le logiciel la connaissance des principes d'organisation du logiciel (modularité, orientation objet, généricité, etc.).

Au troisième niveau il s'agit d'apprendre les motifs récurrents. Pour devenir un maître d'échecs, il faut en effet étudier le jeu des autres maîtres : les parties d'échecs disputées par des grands maîtres contiennent en effet des patrons qui devraient être compris, mémorisés, et appliqués de manière répétitive. De même, l'idée des patrons de conception est que l'expérience de conception logicielle s'acquiert en étudiant des conceptions maîtresses.

2.2 Exemple : le patron observateur

Bien qu'il y ait probablement une gamme infinie de patrons spécifiques de domaines particuliers, il est en revanche probable que la plupart des patrons à usage général qu'un ingénieur logiciel devrait connaître ont déjà été identifiés. L'un des plus connu parmi ceux-ci est probablement *l'observateur*, que nous rappelons ici pour illustrer notre propos.

L'intention du patron Observateur est de définir une dépendance entre un objet et ses multiples observateurs de manière à ce que lorsque cet objet change d'état, tous les observateurs qui en dépendent sont automatiquement notifiés.

Prenons l'exemple d'un serveur de fichiers auquel est connecté un ensemble d'ordinateurs clients (notion de connexion d'un lecteur réseau dans le monde Windows ou de montage d'un système de fichiers dans le monde Unix, voir figure 1).

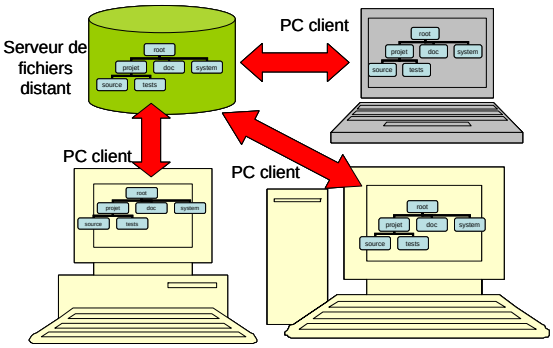

Figure 1. Un système de fichier réparti

À chaque fois qu'un changement se produit sur le serveur (renomage de fichiers ou de répertoires, ajout ou suppression de fichiers), celui-ci doit être reflété sur chaque client de manière à lui permettre de mettre à jour sa vue du système. Une solution à ce problème doit prendre en compte un certain nombre de contraintes qu'on appelle *le contexte* :

- ✓ les identités et le nombre des clients ne sont pas déterminés à l'avance ;
- ✓ de nouvelles sortes de clients peuvent être ajoutées au système dans le futur ;
- ✓ l'interrogation active n'est pas appropriée, car par exemple trop coûteux quand le nombre d'observateurs est grand.

Le patron Observateur résout le problème dans ce contexte en faisant en sorte que le serveur (qu'on appelle le sujet) informe les ordinateurs clients (les observateurs) à chaque fois qu'un changement intéressant se produit. En utilisant un diagramme de classes UML, la figure 2 illustre les relations existant entre un sujet et ses observateurs : il décrit un exemple de structure statique du patron.

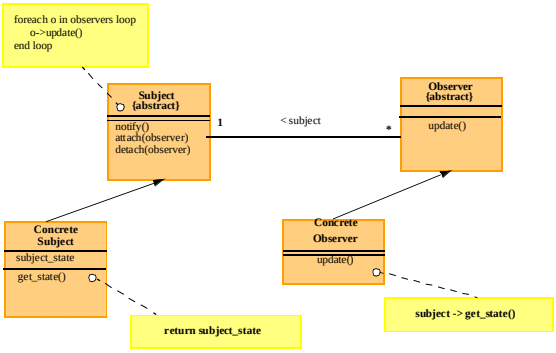

Figure 2. Structure statique du patron Observateur

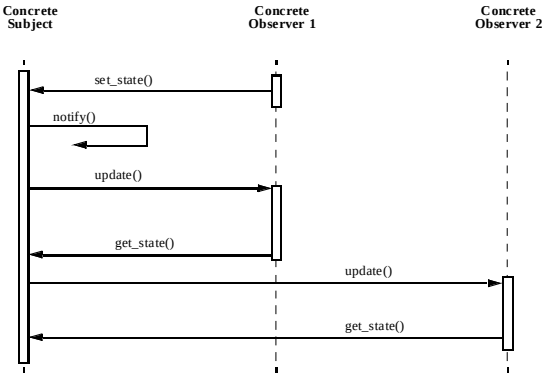

Figure 3. Structure dynamique du patron Observateur

De même la figure 3 illustre l'aspect dynamique d'une collaboration entre un sujet et ses observateurs : c'est un exemple de structure dynamique du patron.

Si vous avez déjà un peu d'expérience en conception de systèmes informatiques, vous devez vous demander pourquoi une solution de conception aussi simple mérite qu'on s'y attarde. Pour répondre à cette question, commençons par prendre un second exemple.

Considérons un outil graphique qui permet à un utilisateur de gérer un ensemble de valeurs de données dans un tableur, un histogramme et un camembert (voir figure 4). N'importe laquelle de ces vues peut être ouverte à n'importe quel moment (y compris plusieurs fois la même vue ouverte simultanément) et visible à l'écran. Chaque fois que l'utilisateur modifie l'une des vues, toutes les autres doivent bien sûr être modifiées en adéquation.

	1er trim.	2e trim.	3e trim.	4e trim.
Est	20,4	27,4	90	20,4
Ouest	30,6	38,6	34,6	31,6
Nord	45,9	46,9	45	43,9

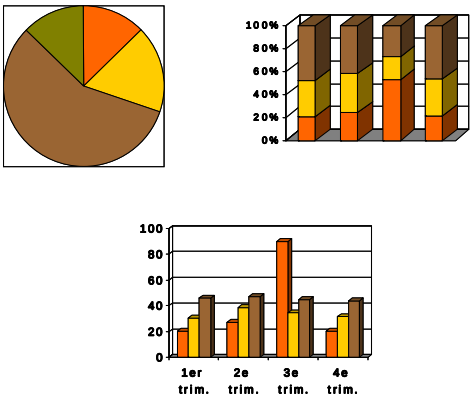

Figure 4. Différentes vues sur un ensemble de données

Remarquons qu'à un certain niveau d'abstraction, nous avons exactement le même problème de conception que pour le serveur de fichiers, avec en particulier les mêmes contraintes :

- ✓ les identités et le nombre des vues ne sont pas déterminés à l'avance ;
- ✓ des nouvelles sortes de vues peuvent être ajoutées au système dans le futur ;
- ✓ l'interrogation active n'est pas appropriée, car par exemple trop coûteuse.

Ainsi la solution peut donc être la même : l'ensemble de valeurs de données joue le rôle du *sujet* alors que les vues jouent celui d'*observateurs* ; ceux-ci interagissant comme illustré aux figures 2 et 3.

Le patron Observateur peut alors être formulé de la manière suivante :

- Objectif : Définir une dépendance de *un vers plusieurs* entre les objets de façon à ce que, quand un objet change d'état, tous ses dépendants sont informés et mis à jour automatiquement ;
- Contraintes clés :
  - il peut y avoir plusieurs observateurs,
  - chaque observateur peut réagir différemment à la même information,
  - le sujet devrait être aussi découplé que possible des observateurs (ajout et suppression dynamique d'observateurs),
- Structure de la solution : voir figures 2 et 3.

Ce qui fait de l'Observateur un patron de conception est qu'il abstrait les architectures statiques et dynamiques d'une solution donnée pour résoudre un problème particulier dans un contexte lui aussi particulier, et qu'il est donc applicable à beaucoup de situations différentes.

2.3 Décrire des patrons de conception

Les patrons de conception sont avant tout des concepts du domaine des idées. Comme toute idée, un patron de conception n'a de valeur que s'il peut être communiqué. Il est donc très important d'avoir un format plus ou moins standard pour pouvoir les décrire. Comme souvent en matière de standards, il en existe plusieurs en pratique. Les plus utilisés sont sans doute le format d'Alexander et celui du GoF. Les patrons d'Alexander sont décrits sous la forme suivante :

- si vous vous trouvez dans un contexte,
- par exemple : *exemple*,
- avec *problème*,
- contraint par *forces*,
- alors pour ces *raisons*,
- appliquer la *forme de conception* et ou *règle*,
- pour construire *solution*,
- amenant à un *nouveau contexte* et d'autres patrons.

Il y a en pratique de nombreuses variations stylistiques à ce format. Celui du GoF inverse en quelque sorte l'ordre de la présentation en commençant

par la forme de la conception puis en décrivant le problème, le contexte et les exemples auxquels elle s'applique. Le canevas de présentation est donc structuré comme suit :

Nom du patron : le nom du patron est choisi de manière à véhiculer succinctement l'essence du patron. Un nom bien choisi est absolument vital, car il devient ensuite un élément clé du vocabulaire de conception. Exemple : *observateur*.

Intention : un court texte ayant pour objectif de répandre la question suivante : que cherche à faire le patron de conception? Quelle est l'intention sous-jacente ? À quel problème particulier de conception s'attaque-t-on ? Exemple : *Définir une dépendance de un vers plusieurs entre les objets de façon à ce que, quand un objet change d'état, tous ses dépendants sont informés et mis à jour automatiquement.*

Alias : autres noms souvent donnés à ce patron de conception (s'il y en a).

Motivations : la motivation ou le contexte dans lequel ce patron s'applique. Ceci peut être concrétisé par un scénario qui illustre un problème de conception et comment une structure de classe et d'objets aide à résoudre le problème. Le scénario a pour objectif d'aider à comprendre les descriptions plus abstraites qui suivront.

Applicabilité : les pré-requis qui doivent être satisfaits pour permettre l'utilisation de ce patron. Quelles sont les situations dans lesquelles ce patron de conception peut être appliqué ? Quels exemples de conception peuvent être résolus par ce patron de conception ? Comment peut-on reconnaître ces situations ?

Structure : une description structurelle (en termes de classes et de leurs relations exprimés par exemple en UML) d'un exemple du type de solution proposée par ce patron. Exemple : voir figure 2. A cet égard, il faut insister sur le fait que le schéma UML décrivant la structure *n'est pas* le patron de conception, mais, au côté des autres rubriques, contribue simplement à véhiculer l'idée constituée par ce patron de conception.

Participants : la liste des participants typiquement impliqués dans l'application de ce patron.

Collaborations : comment les participants collaborent pour mener à bien leurs responsabilités.

Conséquences : les conséquences de l'utilisation de ce patron, tant positives que négatives. Comment le patron de conception supporte-t-il son intention ? Quels sont les compromis et les résultats de l'application du patron ? Quels aspects de la structure du système ne permet-il pas de faire varier de manière indépendante ?

Implantation : quels pièges, astuces, ou techniques concernent l'implantation de ce patron de conception ? Y a-t-il des aspects qui sont spécifiques d'un langage de programmation particulier ?

Exemple de code et utilisation : des fragments de code qui illustrent comment on peut implanter ce patron de conception dans des langages de programmation à objets classiques.

Utilisations connues : exemples documentés d'application de ce patron de conception dans des systèmes réels. L'effort de documentation d'une solution de conception n'est en effet valable que si cette solution peut être appliquée de manière récurrente pour résoudre le problème dans un contexte donné. Il est donc d'usage de considérer qu'une solution de conception n'est en réalité un patron de conception que si elle a été identifiée dans au moins trois applications totalement différentes. C'est bien sûr le cas du patron observateur qui se retrouve d'une manière ou d'une autre au coeur de toutes les interfaces graphiques conçues depuis plus de vingt-cinq ans.

Patrons connexes : quels patrons de conception ont un rapport avec celui qu'on est en train de décrire ? Quelles sont les différences importantes ? Avec quels autres patrons celui-ci peut-il être utilisé en synergie ?

2.4 Patrons de conception et notions connexes

On trouve de manière connexe à la notion de patron de conception d'un côté la notion de patron d'architecture, et de l'autre la notion de patron de codage aussi connu sous le nom d'idiome.

Un patron d'architecture exprime un schéma d'organisation structurelle fondamental pour des systèmes logiciels (Buschmann 1996). Il fournit un ensemble de sous-systèmes prédéfinis en spécifiant leurs responsabilités, et inclut des règles et des guides méthodologiques pour organiser les relations entre ces sous-systèmes. On parlera par exemple du patron d'architecture *trois niveaux (tiers)* pour parler des systèmes constitués par d'une part un serveur

d'application effectuant la majorité des traitements, d'autre part d'un serveur de bases de données et un troisième lieu d'une interface graphique déportée sur une station cliente (voir figure 5).

Figure 5. Patron d'architecture trois tiers

À l'autre extrémité du spectre, on trouve la notion d'idiome. Un idiome peut-être vu comme un patron de bas niveau spécifique à un langage de programmation donné. Un idiome décrit comment implanter des aspects particuliers d'un composant en utilisant les caractéristiques d'un langage de programmation donné (Coplien 1992). Par exemple il peut s'agir de la manière de gérer l'allocation mémoire dans un langage comme C++.

Sur l'échelle de l'abstraction et du niveau de détail, les patrons de conception se situent à un niveau intermédiaire entre ces idiomes (ou patrons de bas niveau) et les patrons d'architecture. Un patron de conception typique se situe au niveau d'un groupe de classes (une demi-douzaine tout au plus) et de leurs relations. Les patrons d'architecture sont des patrons stratégiques de haut niveau qui s'intéressent à des composants large échelle et aux propriétés globales de systèmes logiciels. Les patrons de conception sont des patrons tactiques à échelle moyenne. Ils sont indépendants d'un langage de programmation particulier, même si les patrons les plus utilisés sont fortement biaisés par l'hypothèse que l'on dispose pour l'implantation d'un langage de programmation par objet. En revanche, les idiomes sont des techniques spécifiques d'un langage pour résoudre des problèmes d'implantation de bas niveau.

2.5 Patron de conception et canevas d'application

La notion de canevas d'application (*framework* en anglais) est aussi fortement connexe de celle de patron de conception. En effet un canevas d'application orienté objet est constitué d'un ensemble cohérent de classes interagissant de manière prédéfinie qui peuvent être spécialisées ou instanciées pour réaliser une application. C'est une architecture logicielle réutilisable qui fournit une structure et un comportement générique pour une famille d'applications logicielles dans un domaine particulier.

Un canevas d'application doit être complété par des fonctions spécifiques d'une application donnée afin de former une application complète. Il peut donc être vu comme une application à trous, ou plus précisément comme une structure préfabriquée dans laquelle un ensemble de pièces situées à des endroits spécifiques (appelés points d'accrochage ou encore points chauds) ne sont pas implantés ou bien ont des implantations par défaut qui doivent être redéfinies. Pour obtenir une application complète à partir d'un canevas d'application orientée objet, on doit fournir ces pièces manquantes sous la forme de sous-classes définissant ou redéfinissant les parties manquantes ou existant par défaut, le mécanisme de liaison dynamique (caractéristiques des langages à objet) étant utilisé pour faire en sorte que les éléments prédéfinis du canevas puissent appeler les éléments complémentaires.

Un canevas d'application est ainsi assez différent d'une bibliothèque de classes classiques dans le sens où le flot de contrôle est usuellement bidirectionnel entre les classes spécifiques de l'application et le canevas d'application. Le canevas d'application a fréquemment la charge de gérer la plus grosse part de l'application, le programmeur n'ayant plus idéalement qu'à fournir quelques éléments ici ou là. Mais il arrive souvent que la logique globale de l'application soit assez complexe et que la manière dont doivent collaborer les éléments préexistants et ceux introduits par le programmeur soit non triviale. Les patrons de conception peuvent alors être utilisés pour documenter les collaborations entre les différentes classes du canevas. Symétriquement, un canevas d'application peut mettre en oeuvre plusieurs patrons de conception, certains d'entre eux d'usage général d'autres plus spécifiques. Les patrons de conception et les canevas application sont donc ainsi fortement couplés, mais il n'opèrent pas au même niveau d'abstraction : un canevas d'application est *constitué de logiciel* concret, alors que les patrons de conception sont de la connaissance, de l'information et de l'expérience à *propos de logiciels*. On peut dire que les canevas d'application sont de nature physique alors que les patrons de conception sont d'une nature purement logique (ils sont du domaine des idées) : les canevas d'application sont des réalisations physiques d'une ou plusieurs solutions proposées par des patrons de conception ; les patrons de conception forment en quelque sorte le mode d'emploi pour réaliser ces solutions.

3. Les patrons de conception dans le cycle de vie du logiciel

3.1 Représentation de l'occurrence de patrons de conception en UML

Pour faire face à la complexité toujours croissante des systèmes logiciels, on a de plus en plus souvent recours en informatique comme dans les autres sciences à des techniques de modélisation. La modélisation est en effet l'utilisation d'une représentation en lieu et place d'une chose du monde réel dans un but cognitif. Un modèle remplace un système réel dans un contexte donné, à moindre coût, plus simplement, plus rapidement et sans les risques ou dangers inhérents à une manipulation du monde réel.

Une fois admise l'idée que la modélisation joue un rôle crucial dans le développement de logiciels, il faut encore disposer d'un langage suffisamment abstrait pour être indépendant de telle ou telle plate-forme matérielle ou logicielle, tout en étant suffisamment précis pour permettre de capturer aussi précisément que nécessaire les aspects fondamentaux d'un problème. Depuis les années soixante-dix, où est apparue l'idée qu'il était nécessaire de modéliser avant de programmer, de très nombreux langages et méthodes ont été proposés pour répondre à ce problème. Le standard actuel du domaine est sans aucun doute le langage de modélisation unifiée UML (*Unified Modeling Language*), qui ne fait finalement qu'une synthèse (ou du moins tente de le faire) des meilleures pratiques existantes pour le développement par objets de logiciels.

UML propose des éléments de modélisation et de visualisation pour représenter les différents aspects des systèmes logiciels. Les éléments de modélisation sont généralement manipulés au travers de représentations graphiques (les diagrammes UML), qui contiennent les éléments de visualisation correspondant aux éléments de modélisation.

UML, dans sa version actuelle 2.0, définit treize types de diagrammes, et permet en outre de mélanger différents types de diagrammes, par exemple pour combiner des aspects structurels et comportementaux au sein d'une même représentation.

UML prend en compte en particulier la notion de patron de conception au travers de celle de collaboration paramétrée. Une collaboration est un contexte pour décrire des interactions entre objets. Une collaboration peut donc être utilisée pour spécifier la réalisation de principes de conception. Il est aussi possible de voir une collaboration comme une entité de conception autonome et donc réutilisable. Ceci peut par exemple être utilisé pour identifier l'occurrence de patron de conception dans un modèle de conception. L'application d'un patron de conception peut alors être vue comme une collaboration paramétrée où à chaque utilisation du patron, des éléments de modèles effectifs (comme classes, relations, méthodes, attributs, états, etc.) sont substitués aux paramètres de la définition du patron de conception.

L'utilisation d'un patron de conception est représentée comme une ellipse en pointillé contenant le nom du patron de conception. Une ligne pointillée est tirée entre cette ellipse et chacune des classes (ou chacun des objets s'il s'agit d'un diagramme objet) participant à cette collaboration. Chacune de ces lignes est étiquetée avec le nom du rôle joué par le participant dans la collaboration. Le rôle correspond au nom des éléments décrits dans le contexte de la collaboration (par exemple pour le patron *observateur*, on trouve les rôles *sujet* et *observateur* qui doivent être joués par des classes, mais aussi les rôles *notify* et *update* qui doivent être joués par des méthodes définies dans les classes jouant les rôles sujet et d'observateurs). Ces noms de rôle peuvent être vus comme des paramètres formels, alors que les éléments du modèle de l'utilisateur sont des paramètres effectifs, les liaisons entre paramètres formels et paramètres effectifs se faisant graphiquement comme illustré dans la figure 6.
Cette notation à l'avantage de rendre explicite l'utilisation d'un patron de conception donné dans un modèle de conception sans avoir à entrer dans les détails de son implantation.

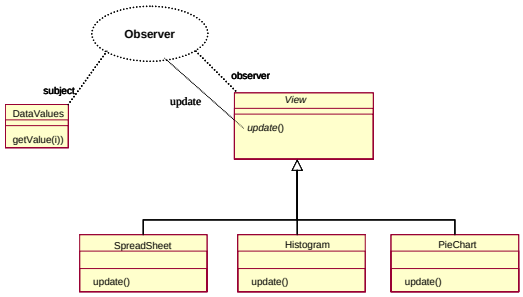

Figure 6. Utilisation du patron observateur en UML

3.2 Des problèmes aux solutions avec les patrons de conception

L'activité de conception par objets de logiciels a pour objectif fondamental de prendre en compte les forces non fonctionnelles (telles que fiabilité, sécurité, performance, ponctualité, consommation d'énergie, qualifié de service, flexibilité, évolutivité, maintenabilité, etc.) implicites dans un modèle d'analyse pour aller vers un modèle qui sera directement implantable dans un langage à objets donné. L'objectif d'une analyse par objet est de modéliser le domaine du problème de façon à ce qu'il puisse être parfaitement compris et servir de base stable à la conception. Le processus de conception peut être vu comme une transformation progressive du modèle d'analyse en un modèle d'implantation par l'application successive d'un ensemble de règles de conceptions. Idéalement, ces règles de conceptions devraient toutes pouvoir être exprimées comme des patrons de conception. On parle alors de *langage de patrons* (*pattern language*), dans le sens où chaque patron de conception individuel joue le rôle d'un mot dans un vocabulaire, et où leur articulation cohérente forme des phrases faisant sens et permettant de résoudre des problèmes globaux de conception. Vlissides a par exemple montré dans (Vlissides 1998) comment l'articulation de cinq des patrons de conception du GoF permettait de concevoir un système de fichiers :

- Composite : pour fournir la structure arborescente du système de fichiers ;
- Proxy : pour supporter la notion de lien symbolique (ou de raccourci) ;
- Template Method : pour fournir une protection d'accès aux données ;
- Singleton : pour fournir deux niveaux de protection ;
- Mediator : pour supporter la notion de groupe d'utilisateur pour l'accès aux fichiers.

Dans ce contexte de transformation progressive du modèle d'analyse en un modèle d'implantation par l'application successive d'un ensemble de règles de conceptions, UML peut être utilisé comme l'épine dorsale supportant ce processus en permettant au concepteur de progressivement entrer dans les détails de son implantation sans changer de langage de modélisation.

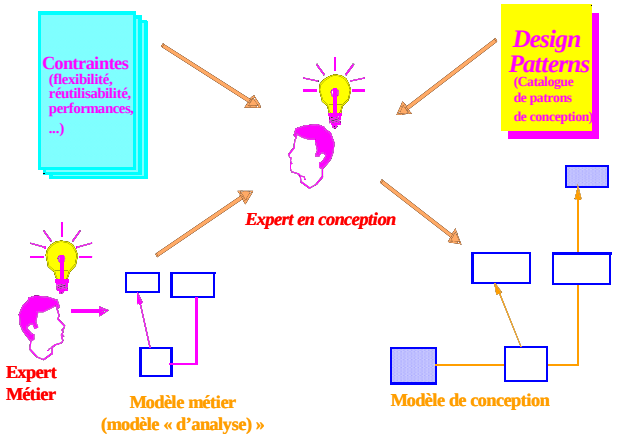

Figure 7. Le rôle du concepteur

3.3 Vers l'automatisation de l'application de patrons de conception

Le statut des modèles évolue d'une phase « contemplative » à une phase « productive » : auparavant les modèles étaient considérés comme des éléments de documentation ; ils sont maintenant de plus en plus souvent devenus des entités de première classe, traités comme entrées ou sorties par des processeurs automatiques. Ceci permet entre autres d'envisager l'application automatique de patrons de conception.

Bien sûr, ce qui fait la difficulté de la chose est que les solutions de conception proposées par un patron de conception particulier diffèrent presque toujours dans leurs détails tout en restant semblables dans leurs principes. De plus, un patron de conception n'est par principe jamais « complet », dans la mesure où il doit permettre une infinité de variation autour d'un même thème. Cependant, si la reconnaissance des problèmes de conception et le choix d'une variante particulière de l'application d'un design pattern restent des activités difficilement mécanisables (sauf techniques d'intelligence artificielle), il n'en est pas de même pour la phase d'application de la solution de conception.

Depuis (LeGuennec 2000) on dispose en effet d'un formalisme de description des patrons de conception au niveau du métamodèle UML, suffisamment précis pour être exploitable par un outil d'application automatique (France 2004). Par simple transformation de modèle, il est ainsi possible de passer automatiquement d'un modèle de conception où l'on mentionne qu'un patron de conception doit être utilisé (par exemple le patron de conception *Command* pour la figure 8) à un modèle de conception montrant le résultat de l'application d'une variante particulière de ce patron de conception (voir figure 9).

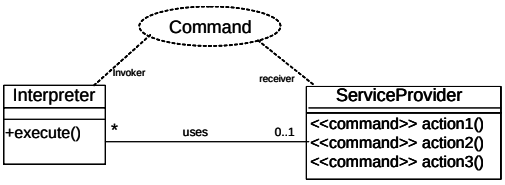

Figure 8. Décision d'utiliser un patron de conception

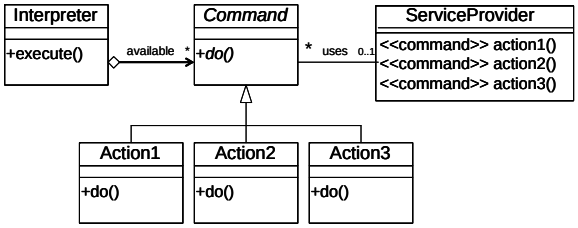

Figure 9. Résultat de la transformation automatique de modèle

La difficulté est alors de fournir de telles transformations de modèles de manière à préserver l'extensibilité du jeu de solutions proposé par un patron de conception donné. Pour cela, deux approches sont habituellement envisagées, l'une fondée sur la programmation logique, qui par sa nature déclarative ne limite pas *a priori* l'ensemble des solutions possibles. L'autre, probablement plus répandue, consiste à utiliser pour la conception de ces transformations de modèles les mêmes principes de conception par objets qui président à la conception de l'application : principe d'ouverture-fermeture modulaire apporté par l'héritage et la liaison dynamique (Meyer 1997), et patrons de conception permettant l'extensibilité des solutions proposées. Les patrons de conception sont alors utilisés en quelque sorte réflexivement pour réaliser la flexibilité nécessaire à l'implantation des transformations de modèles permettant l'application de patrons de conception.

En pratique au niveau industriel, il existe aujourd'hui de nombreux prototypes d'outils qui automatisent plus ou moins cette application de patrons de conception. Il faut bien avouer cependant que la plupart d'entre eux ont encore une certaine marge de progression en ce qui concerne la flexibilité et le paramétrage des solutions de conception proposées aux concepteurs.

4. Conclusion

La généralisation dans l'industrie d'une approche par objets de la conception de logiciel a permis de focaliser cette tâche de conception sur la description des schémas (ou motifs) d'interactions entre objets, et d'assigner des responsabilités à des objets individuels de telle sorte que le système résultant de leur composition ait les propriétés extra-fonctionnelles voulues. L'aspect récurrent de certains de ces schémas d'interactions entre objets a conduit à vouloir les cataloguer sous la forme de *design patterns* ou patrons de conception.

Au cours de dernière décennie, on est ainsi passé progressivement d'une approche artisanale et approximative de la conception à une application rationnelle de solutions ayant fait l'objet d'un catalogage systématique parce que issues des meilleures pratiques du domaine. Aujourd'hui, on aborde un nouveau défi concernant l'automatisation de l'application de ces solutions de conception à l'aide des notions de transformation de modèle ou de tissage d'aspects. D'abord cantonnée au niveau de la programmation, la notion de tissage aspects remonte au niveau des modèles de conception et rejoint celle de transformation de modèles en s'appuyant sur le fait que dans le domaine du logiciel, un modèle a la particularité d'avoir la même nature que la chose qu'il modélise, ce qui ouvre la possibilité de dériver automatiquement, depuis un modèle, une conception logicielle, ainsi d'ailleurs que d'autres artéfacts comme le code, les cas de test, des profils de performances, ou de la documentation. Le défi reste bien sûr toujours de concilier automatisation et flexibilité afin de ne point restreindre la créativité des concepteurs.

Bibliographie

(Alexander, 1979) Alexander, Christopher *The Timeless Way of Building*, NY: Oxford University Press, 1979.

(Buschmann, 1996) F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern Oriented Software Architecture, A System of Patterns*, Wiley & Sons, 1996

(Coplien, 1992) J.O. Coplien, *Advanced C++ Programming Styles and Idioms*. MA: Addison-Wesley, 1992.

(Coplien, 1995) J.O. Coplien and D.C. Schmidt (eds), *Pattern Languages of Program Design*. MA: Addison-Wesley, 1995.

(Coplien, 20005) J.O. Coplien, *Organizational Patterns: Beyond Technology to People*. MA: Addison-Wesley, 2005.

(Demeyer, 2003) S. Demeyer, S. Ducasse, O. Nierstrasz *Object-oriented reengineering patterns*. Morgan Kaufmann, 2003.

(France, 2004) R.B. France, D-K. Kim, S. Ghosh, and E. Song, A UML-Based Pattern Specification Technique. *IEEE Trans. Software Engineering*, vol. 20, no. 3, pp. 193-206, Mar. 2004.

(Gamma, 1995) E. Gamma, R. Helm, R. Johnson, and J. Vilissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. MA: Addison-Wesley, 1995.

(Harrison, 2000) N. Harrison, B. Foote, H. Rohnert, Eds. *Pattern Languages of Program Design 4*. MA: Addison-Wesley, 2000.

(Jézéquel, 1999) Jézéquel, Jean-Marc, *Design Patterns and Contracts*, Addison-Wesley, 1999. ISBN 0-201-30959-9

(Le Guennec, 2000) A. Le Guennec, G. Sunyé, and J.-M. Jézéquel, « Precise modeling of design patterns». - In *Proc. UML2000*, Springer, LNCS 1939.

(Martin, 1998) R. Martin, D. Riehle, F. Buschmann, Eds. *Pattern Languages of Program Design 3*. MA: Addison-Wesley, 1998.

(Meyer, 1997) B. Meyer, *Object-Oriented Software Construction*. 2nd Ed. NJ: Prentice Hall, 1997.

(Reenskaug, 1996) T. Reenskaug, P. Wold, and O.A. Lehne, *Working with Objects, The OOram Software Engineering Method*, Greenwich: Manning Publications Co, 1996.

(Tichy, 2003) W.F. Tichy, *A Catalogue of General-Purpose Software Design Patterns*, Available at http://wwwipd.ira.uka.de/~tichy/publications/Catalogue.doc

(Vlissides, 1996) J. Vlissides, J.O. Coplien, and N. Kerth eds. *Pattern Languages of Program Design 2*. MA: Addison-Wesley, 1996.

(Vlissides, 1998) J. Vilissides, *Pattern Hatching: Design Patterns Applied*. MA: Addison-Wesley, 1998.

(Wirfs-Brock, 1990) R. Wirfs-Brook, B. Wilkerson and L. Wiener, *Designing Object-Oriented Software*. NJ: Prentice Hall, 1990.

(WWWPortal) http://hillside.net/patterns

(Zhao, 2003) L. Zhao and J.O. Coplien, Understanding Symmetry in Object-Oriented Design and Programming, *Journal of Object Technology*, September 2003. Available: http://www.jot.fm